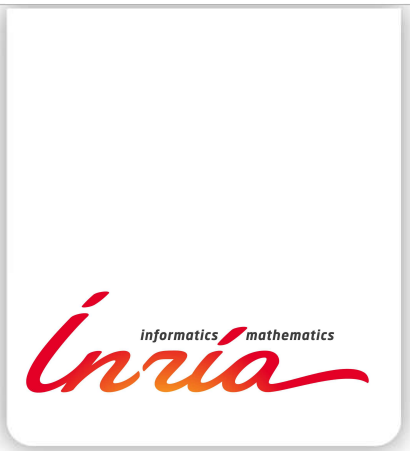
Discrete Voronoi-like Partition of a Mesh on a Cellular Automaton in Asynchronous Calculus

Nassim Kaldé, Olivier Simonin

RESEARCH REPORT
N° 8547
Juin 2014
Project-Teams MAIA

ISSN 0249-6399 ISRN INRIA/RR--8547--FR+ENG



Discrete Voronoi-like Partition of a Mesh on a Cellular Automaton in Asynchronous Calculus

Nassim Kaldé, Olivier Simonin

Project-Teams MAIA

Research Report n° 8547 — Juin 2014 — 20 pages

Abstract: This report presents an approach for asynchronously computing a Voronoi skeleton in a decentralized fashion on a regular grid of cells with a von Neumann neighborhood. To our knowledge, no previous work asynchronously solves this problem in a decentralized fashion. The methods given in this paper describe algorithms for extracting an area voronoi diagram skeleton and a pseudo line-like voronoi diagram skeleton. Algorithms are implemented in simulation and executed on maps in which we consider different kind of sources defined as polygonal shapes. Such work can be useful for robotic navigation.

Key-words: asynchronous computing, cellular automata, ambient intelligence, distributed systems, spatial computing, voronoi diagram

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Partition Discrète de Voronoi sur une Grille d'Automate Cellulaire par Calcul Asynchrone

Résumé : Ce rapport présente une approche pour le calcul asynchrone et décentralisé d'un squelette de Voronoi sur une grille régulière de cellules possédant un voisinage de von Neumann. À notre connaissance, aucun travail précédent ne résout ce problème de manière asynchrone et décentralisée. Les méthodes données dans ce papier décrivent des algorithmes d'extraction du squelette d'un diagramme de voronoi d'aire ou de ligne. Ces algorithmes ont été implémentés en simulation et éxécutés sur des cartes dans lesquelles nous avons considéré différentes sources de forme polygonale. Ce travail peut être utile en navigation robotique.

Mots-clés : calcul asynchrone, automate cellulaire, intelligence ambiante, systèmes distribués, calcul spatial, diagramme de voronoi

1 Introduction

We are interested in computing Voronoi like partition of the environment using an asynchronous mesh of processing and sensing units. Ambient intelligence [4] relies on numerous processing units located and communicating in a same environment. Designing algorithms for self-organizing systems that can handle communication between the units, are scalable and simply work in such environments is part of the domain of spatial computing [5] where the data is distributed in the network and the computing can be done in various synchronization modes.

The environment on which we focus is a typical lattice of a cellular automaton (CA), each cell has 4 direct neighbors. A common formalization of this lattice is given in (1). A 2D CA is a tuple $\mathbb{F} =< A, Q, u_1, f >$, with u_1 being the von-Neumann neighborhood, i.e. a mapping that associates each cell c_{ij} in A, to itself and its North, East, South and West neighbors.

$$\begin{aligned}
&\textbf{2D Cellular Automaton : } \mathbb{F} \\
&\mathbb{F} =< A, Q, u_1, f > \\
&A : a\ matrix\ of\ cells\ (dimension\ m \times n) \\
&Q : a\ set\ of\ states\ of\ a\ cell \\
&u_1 : a\ cellular\ neighborhood\ A \rightarrow A^5 \\
&f : a\ local\ rule\ of\ evolution\ Q^5 \rightarrow Q
\end{aligned} \tag{1}$$

The automaton evolves in discrete time by applying a global rule of evolution F on its matrix of cells A, as shown in (2) and (3).

$$\mathbb{F}^t =< A, Q, u_1, f >^t =< A^t, Q, u_1, f > \tag{2}$$

$$\mathbb{F}^{t+1} =< F(A^t), Q, u_1, f >=< A^{t+1}, Q, u_1, f > \tag{3}$$

In this paper, we study CA which are not synchronized. We consider that the automaton evolves at each step by randomly selecting an individual cell on the grid (4) and executes its local evolution rule, cf. (5). We call this evolution mode, the *simple asyncronous mode*. Only one cell c updates its state at $t+1$ by applying the local rule of evolution f on its neighborhood at time t.

$$A^{t+1} = \{c \in A | \exists! c \in A : c^{t+1} \neq c^t\} \tag{4}$$

$$c^{t+1} = f(u_1^t(c)) \tag{5}$$

Here, we address the problem of asynchronously computing Voronoi-like diagrams using a CA. Several papers already cover the problem of computing Voronoi tesselations or Voronoi skeleton synchronously. This work presents the consequences of asynchronous computing on these approaches, showing how they fail. Then we propose methods to obtain similar results in *simple asynchronous mode*. We tackle the problem by adapting some synchronous methods to asynchronous computation.

Section II of this report exposes some works related to computation of Voronoi diagrams and skeletonization. The third section shows the limitation of the basic approaches to solve the problem in the simple asynchronous mode. Section IV is our contribution, it explains how we can adapt some prior works to this challenge. Then, section V evaluates this work in simulation. We conclude by giving a short table that presents the results of the different methods and our method depending on different criteria.

2 Related Work

Voronoi partition of lattices and Voronoi skeleton extraction have given birth to numerous studies since the 17th century when René Descartes fragmented space maps into Voronoi-like regions. Skeletonization was investigated as a new shape descriptor by Blum in [3], he introduced the well-know fire-propagation on a grass field metaphor as a model to compute the skeleton of a shape. We give the common definitions of the discrete generalized Voronoi diagram (6) and skeleton of a contour (7) as a reminder.

The Voronoi discrete diagram VD is a set of Voronoi regions $VR(S_i)$, it represents a tesselation of the matrix of cells A. The region $VR(S_i)$ of a seed area S_i is formed by the cells c_{kl} which are closer to this seed area than any other seed area.

$$\begin{aligned} &\textbf{Generalized Voronoi Diagram : } VD \\ &VD = \{VR(S_1), ..., VR(S_n)\} \\ &VR(S_i) = \{c_{kl}/d(c_{kl}, S_i) \leq d(c_{kl}, S_j), \forall j \neq i\} \\ &S_i \subset A, 0 < k \leq m, 0 < l \leq n, d(a, b) \textit{ a metric} \end{aligned} \tag{6}$$

The Voronoi discrete skeleton VS of a contour C is the set of cells c_{ij} which are at the ridges in distance map of the cells c_{kl} supporting C, see [8]. These skeleton cells can be defined as the breaking cells of a synchronous wave propagated from the contour (7). In other words, they are the loci where at least two wave fronts meet.

$$\begin{aligned} &\textbf{Voronoi Skeleton : } VS \\ &VS = \{c_{ij}/|\{c_{kl} \in C/d(c_{ij}, c_{kl}) = w(c_{ij})\}| \geq 2\} \\ &C \subset A, 0 < i, k \leq m, 0 < j, l \leq n \\ &w(p) \textit{ distance to } C \end{aligned} \tag{7}$$

In the domain of CA, the techniques used for computing the Voronoi diagram or skeleton are synchronous. We begin by introducing two common approaches: the first one is based on the fire propagation metaphor [1], and the second one is known as wavefront expansion [2].

The principle of the former approach is quite simple: the fire starts its propagation from the boundaries of fire sources and travels to the next neighbors synchronously. When two fire fronts meet on a same cell, this cell is a part of the Voronoi skeleton. The second approach is quite similar as it propagates a front, which is a wave growing from the boundaries of the wave sources. At each step the wave value increases. And when two waves collide on the same cell, this cell is in the Voronoi skeleton. To abstract, the fire is unaltered while crossing the environment (constant wave) whereas the wave grows steadily during its expansion.

To our knowledge, there is no reference to Voronoi skeleton extraction or Voronoi diagram computed in an asynchronous mode. We focus on Voronoi skeleton extraction in the rest of this paper because we are interested in exploiting this structure as roads for robotic navigation on discretized floor in future works. The Voronoi skeleton can be formulated as a set of bisectors between seed areas. Thus, the basic case of Skeleton extraction is the bisector extraction between two isolated seeds.

Next section introduces the definition of discrete bisector and gives a few more details about two common synchronous approaches and how they fail to extract bisectors asynchronously.

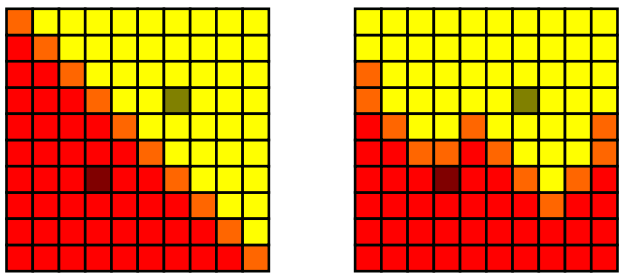
3 Limitation of Syncronous Computation

In this part we will focus on *bisector extraction* and explain why the presented techniques do not work for asynchronous computing. We examine two approaches by providing the expected bisector and the computed bisector to discuss its validity.

The common mathematical definition of a bisector is not adapted to discrete computation. We refer to the discrete definitions given in [1]. The bisector Bis of cells a and b is defined as the set of cells c such that the distances, from c to a and c to b, differ by at most 1, formally in (8). The distance used here is the Manhattan distance d_1.

$$\begin{aligned} &Bis(a,b) = \{c \in A : |d_1(a,c) - d_1(b,c)| \leq 1\} \\ &d_1(c_{ij}, c_{kl}) = |k-i| + |l-j| \end{aligned} \tag{8}$$

3.1 Fire Propagation with Identifier



(a) synchronous mode (b) asynchronous mode

Figure 1: Identifier map computing with two seeds (dark red/dark green), two Voronoi regions (red/yellow), one discrete bisector (orange)

3.1.1 Principle

The principle applied here is the following: each seed cell has a unique identifier and this identifier is synchronously propagated to the neighboring cells on the lattice. When a cell receives an identifier, it stores this information forever, disabling any further update. Cells receiving two different identifiers at the same time are exactly equidistant to two seeds separated by an odd distance. Cells receiving only one identifier but located at the boundaries of an identified region are also considered equidistant (modulo 1) to two seeds separated by an even distance. In synchronous calculus, these cells support the bisector between the two seeds.

3.1.2 Synchronous Fire

Figure 1.a shows an example of an *expected identifier map*, computed synchronously following this principle. Cells colored in *yellow/red* have stored the identifier coming from the *dark green/dark red* source. This map represents two regions of a Voronoi diagram, and the bisector between these regions is in *orange*. Here the seeds are separated by an odd distance. Thus, simultaneously receiving different identifiers is sufficient to determine that a cell is on the bisector.

Thanks to the synchronism, a simple CA with 4-state cells can compute this bisector, details are available in [1].

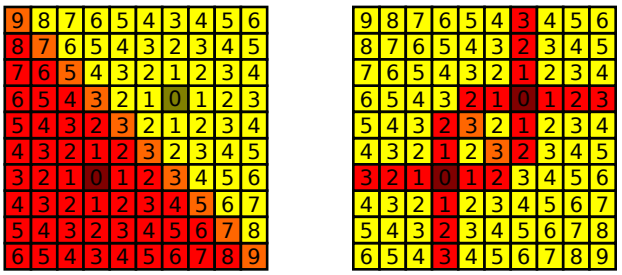
3.1.3 Limitation

It is clear that this principle of identifier propagation is not wise in asynchronous computing. The underlying assumption, in synchronous calculus, is that the computed frontier between identified regions is the effective bisector between the seeds. But, by randomly updating a unique cell at each step, this assumption does not hold true. And the *computed identifier map* will probably not converge to the *expected identifier map*.

Figure 1.b. presents a *computed identifier map*, obtained by applying the fire principle asynchronously. By randomly selecting a cell to update on the lattice at each step, we can propagate identifiers and fix them onto the cells. The result illustrates a fire propagation influenced by random wind variation, not a fire constant propagation.

The regions in red and yellow are not the correct Voronoi regions, neither has the extracted frontier converged to the expected bisector.

3.2 Wavefront Expansion with Distance Map



(a) synchronous mode

(b) asynchronous mode, colors are relief patterns

Figure 2: Distance map computing with two seeds (dark red/dark green), one discrete bisector (orange)

3.2.1 Principle

The method exposed now is based on a distance map. The distance map used is a d_1 *map* on which each cell locally computes its Manhattan distance to the closest seed(s), see Fig.2.a. It can be computed synchronously by the wavefront expansion algorithm presented in [2]. The procedure is divided into the following steps:

1. Non-seed cells are set to infinity
2. Seed cells are set to 0 to start the wave propagation
3. A gradient is built by increasing the values of the nearest yet unreached cells

During this procedure, a list of currently considered cells is maintained, representing the wavefront. Cells on which the fronts collide belong to the bisector according to a threshold on the distance between the identified seeds.

3.2.2 Synchronous Wave

Figure 2.a illustrates the expected result of a 9-step synchronous diffusion from two seeds. In *red/yellow* the Voronoi region of the *dark red/dark green* source is shown. Cells colored in *orange* represent the bisector between the two seeds.

3.2.3 Limitation

First of all, the idea of extracting a bisector by selecting front meeting cells does not stand in *simple asynchronous mode*. The reason is still that asynchronous fronts can possibly meet on cells which do not support the bisector.

Nevertheless, computing a distance map can be achieved asynchronously. The asynchronous gradient rule G that provides such a distance map is expressed in (9). It simply updates the distance of a cell c to its closest seed(s) by adding one to the minimum distance computed by its neighbors. The *expected distance map* is provided as soon as the rule has converged.

$$\begin{aligned} &G(c) : c.d_1 \leftarrow \begin{cases} \text{m+1}, & if\ c\ not\ seed \\ 0, & else. \end{cases} \\ &m = min_{n_i \in u_1(c)}(n_i.d_1) \end{aligned} \tag{9}$$

But, when we try to exploit this gradient to identify front meeting cells, it is not sufficient to isolate a full bisector, see Fig. 2.b. Each cell has a local knowledge of the gradient variation in its u_1 neighborhood. We classify these different gradient patterns into 4 different rotation invariant classes of relief. Therefore, seeds in *dark red* are local minima, cells in *orange* are local maxima, and cells in *red* are unidirectional minima. In this counterexample, when it comes to bisector extraction, only the local maxima class is informative (*orange*). Here, the two micro-sources have created a kind of macro-source that emits its own wave. The two asynchronous waves merge and do not collide, we call this phenomenon the waves fusion.

We can now state that for the simple asynchronous evolution, firefronts are subject to random wind variation and wavefront collisions are sometimes not detectable due to the phenomenon of waves fusion. Next section provides a working asynchronous method for computing identifier maps. It is based on the idea that using an convergent asynchronous gradient computation enables to build an identifier map at the same time. Thus a true identifier map can be computed asynchronously as soon as the gradient converges to the real distance map.

4 Our Contribution

First, this section presents a method for asynchronously computing an identifier map in a decentralized fashion. It requires to build a gradient at the same time and provides a basic Voronoi tesselation. Then, we introduce a rule to asynchronously extract a Voronoi skeleton as the set of bisectors between the computed Voronoi regions.

4.1 Identifier Map and Convergent Gradient

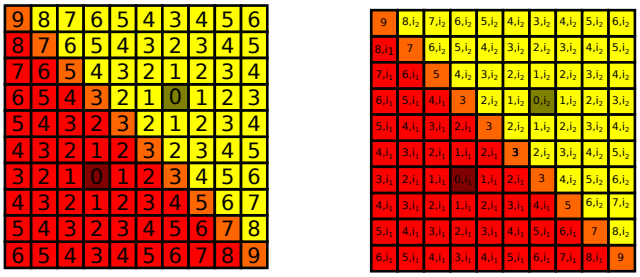
To overcome the weaknesses (random wind, waves fusion) noticed earlier, we design a procedure that allows to compute a correct identifier map in asynchronous calculus.

4.1.1 Asynchronous propagation rule

For this, we use both the *gradient* (d_1) and *identifier* (id) as a couple of information for each cell c. Each cell updates the received identifier until the gradient has converged in the whole network. This couple is asynchronously propagated from the sources onto the lattice of cells using the new rule Pr (10).

$$
\begin{aligned}
&Pr(c) : (c.d_1, c.id) \leftarrow \begin{cases} (\mathrm{m}{+}1, u), \; if \; c \; not \; seed \\ (0, c.id), \quad else. \end{cases} \\
&m = min_{n_i \in u_1(c)}(n_i.d_1) \\
&u = \{k.id | k \in u_1(c), k.d_1 = m\}
\end{aligned}
\tag{10}
$$

4.1.2 Asynchronous propagation example



(a) synchronous expected bisector

(b) asynhronous computed bisector

Figure 3: Distance-gradient only in synchronous mode (a) and Distance-Identifier mix in asynchronous mode (b)

By combining both previous synchronous ideas into rule Pr, we obtain a correct asynchronously *computed identifier map* in Figure 3.b. To fix the good identifiers in the cells, we wait until the d_1 map has converged. After convergence, a Voronoi region ($red/yellow$) is correctly identified as a set of cells owning the same seed identifier.

The bisector is then extracted as the frontier between two Voronoi regions. This simple mechanism of bisector extraction is detailed next as we use it to extract skeleton cells.

4.2 Skeleton extraction

Skeleton extraction is achieved by marking the bisectors of the Voronoi regions. In the following, we introduce a rule to allow local asynchronous emergence of the skeleton cells.

4.2.1 Skeleton extraction rules

An intuitive way to extract the frontier between two Voronoi regions is by marking the boundaries of these regions. Skeleton extraction works similarly, if we define the skeleton as the set of frontiers between Voronoi regions. In a cell centered view, a skeleton cell owns an identifier which is different from the identifier of at least one of its neighbors. This first rule Sk formalizes this intuition in (11), but the extracted skeleton is thick.

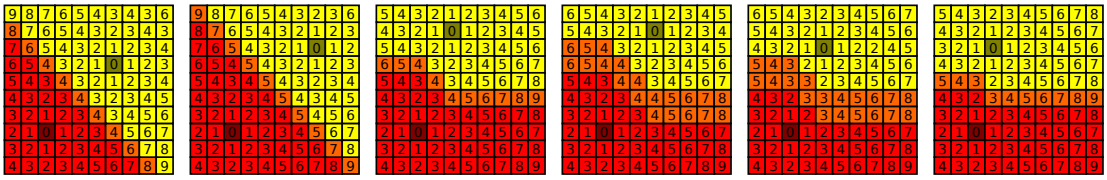
$$Sk(c) : c.sk \leftarrow \begin{cases} 1, \, if \ (\exists n \in u_1(c)/n.id \neq c.id) \\ 0, \, else. \end{cases} \tag{11}$$

Therefore, we design another rule for thinner skeleton extraction in (12). This rule is based on the discrete bisector definition, two types of bisector cells exist. The first type corresponds to cells wich are exactly equidistant to two sources, and the second one represents (modulo 1) equidistance to sources. From a cell centered view, first type skeleton cells receive exactly 2 identifiers (set *skel* to 1). Second type skeleton cells receive one or less identifier and have a neighbor which owns only one but different identifer (set *skel* to 2).

$$Sk(c) : c.sk \leftarrow \begin{cases} 1, \, if \ |c.id| > 1 \\ 2, \, if \ |c.id| \leq 1 \wedge (\exists n \in u_1(c)/|n.id| = 1 \wedge n.id \neq c.id) \\ 0, \, else. \end{cases} \tag{12}$$

Next, we show the results for all cases of two simple seeds bisector extraction using Rule Sk (12).

4.2.2 Examples



(a) case 1 (b) case 2 (c) case 3 (d) case 4 (e) case 5 (f) case 6

Figure 4: Bissector Extraction Cases for two seeds a (red) and b (green)

In order to exhibit each kind of possible bisector we reproduce the examples from [1]. Six cases are generated by varying the spacing between two simple seeds according to odd or even d_1-distance, horizontally or vertically. The thin bisectors extracted using (12) are given in Figure 4. Cases 1, 2, 3 and 6 are for odd distances between the seeds, the bisector has a thickness of one cell. For the cases 4 and 5, the bisector has a thickness of two cells because the distance between the seeds is even.

5 Application

5.1 Area Voronoi Diagram

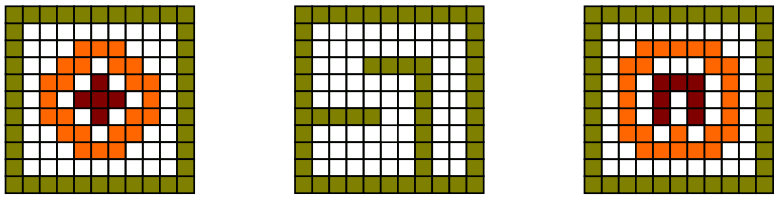
We can already compute a basic area Voronoi diagram [6], without synchronization, by identifiying seeding areas instead of isolated source cells, see Fig.5.

5.1.1 Area Identification

Source areas emerge by aggregating contiguous source cells during a phase of *area identification consensus*. The area identifier combines each aggregated cell identifier, using rule *Aid* presented in (13). This local rule allows two contiguous sources to update their identifier by combining the previous ones. After a certain amount of time, every cell in a same contiguous area share the same identifier.

$$Aid(c,d) : c.id \leftarrow \begin{cases} c.id \cup d.id, & if\ c,d\ seeds \\ c.id, & else. \end{cases} \tag{13}$$

5.1.2 Example



(a) two areas (b) maze (c) concavity

Figure 5: Area Voronoi diagrams skeleton extractions

Identifier based skeleton extraction works in simple asynchronous mode for an area Voronoi diagram. Each cell executes 3 steps: the first one is the *area identification*, which is followed by the *identifier propagation*; and finally the *skeleton extraction* itself (resp. by the rules *Aid*, *Pr*, *Sk*. For example, Fig. 5.a represents two source areas (*green* and *red*) and a skeleton in *orange*. For the simple maze in Fig. 5.b, no skeleton is extracted because the seeds are all contiguous and represent a single area (*green*). In Fig. 5.c, an *orange* skeleton is extracted between the two seeding areas (in *green* and *red*).

5.1.3 Limitation for robotic navigation

These examples bring forward the fact that the skeleton extraction of an area Voronoi diagram is not satisfying for establishing maximum-clearance roadmaps useful in robotic navigation. In the context of robotic navigation on the area Voronoi skeleton, a robot could not navigate into the *light green* maze (Fig. 5b), neither enter the *dark red* concavity (Fig. 5c). These issues are directly related to the definition of the area Voronoi diagram. More complete skeletons can be extrated by considering the line Voronoi diagram.

5.2 Line Voronoi Diagram

In this section, we show how to compute a line-like Voronoi diagram [6] and extract the associated skeleton, see Fig.7. First, the idea is to generate inner bisectors between the edges of a shape for obtaining a more complete skeleton. Thus we discuss *edge identification* which requires to isolate the edges located on the contour of a diffusion area. Then we illustrate the mechanisms of edge identification consensus and line voronoi skeleton extraction.

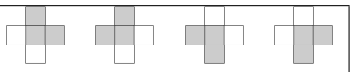
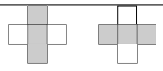
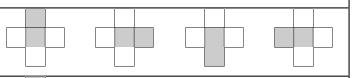
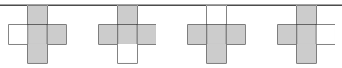
5.2.1 Edge Identification

In our approach we consider an identifier for each side or face of an area as studied earlier in [9]. Therefore, each area of diffusion now creates multiple identifiers. As an hypothesis of simplification, we decide to identify horizontal, vertical and diagonal contour lines only. So the model presented here will only work for sources with diagonal, vertical or horizontal faces.

Classes and Local Patterns Each cell uses a local mask m_i to identify its edge type, the set of masks M is given in Table 1. These masks are classified into 6 classes of rotational symmetry. And elements in a same class share the same behavior for face identification. For the sake of simplicity we design the function $Class_i$ which maps the cell to its edge class, see (14).

$$Cl(c) : c.cl \leftarrow \begin{cases} Class_i(c, M) & if\ c\ seed \\ 0 & else. \end{cases} \tag{14}$$

Table 1: Classes and source patterns

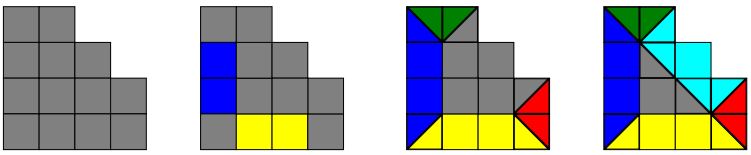
I Isolated		II Corners	
III Lines		IV Extremities	
V Directions		VI Full	

Edge Identification This procedure is called an *edge identification consensus*, see (15). It is a consensus-decision-making which requires communications and delay before every cell on the edge agree on a common identifier. Detailed consensus rules are given in appendix. It is important to notice that $Class_{II}$ and $Class_{VI}$ cells own multiple identifiers (up to 4) depending on the direction of propagation.

$$Eid(c, d) : c.id \leftarrow \begin{cases} Eid_i(c, d), & if\ c.cl = Class_i \\ c.id, & else. \end{cases} \tag{15}$$

5.2.2 Example

To recapitulate, when one cell is selected, it executes the 3 following steps : *edge identification*, *identifier propagation* and *skeleton extraction*.



(a) source area (b) Eid_V (c) Eid_{II} (d) Eid_{VI}

Figure 6: Edge identification process

Edge identification To illustrate how the *Eid* rule works, we identify edges step by step on an example of a source area, presented in Fig.6:

a) A gray polygonal shaped source area which will emit multiple identifiers onto the lattice.

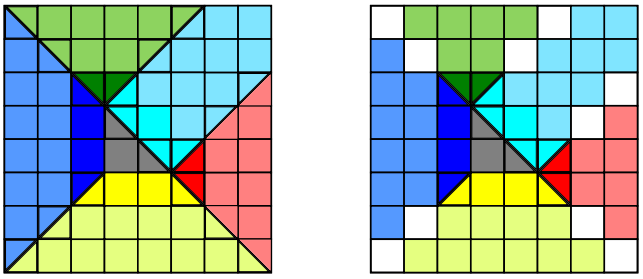
b) Blue and yellow sub-areas appear on the contour. Each color represents an edge identified by Eid_V for $Class_V$ cells. Cells with a same color now possess the same identifier.

c) Blue and yellow sub-areas have expanded. New edges in Green and red were identified. Some *corner* cells now possess two identifiers for belonging to two perpendicular edges.

d) A last color (aqua) appears to identify a diagonal edge. *Corners* and *full* cells worked together to identify this line. $Class_{VI}$ cells act as intermediates between *corners* that cannot communicate in u_1 neighborhood.

Identifier propagation and skeleton extraction The extracted skeleton is evolving dynamically according to the distance-identifier map. It is correct as soons as the asynchronously computed map converges. For our example, the map and skeleton are given in Fig.7.

In Fig.7a, the edges are already colored according to the previous step of *identification consensus*. Now these edges act as sources and the free cells around them are colored according to the computed distance-identifier map. This map is computed by using Pr again (10). Fig.7b presents the skeleton extracted using (12)



(a) Multiple diffusion (b) Skeleton extraction

Figure 7: Line Voronoi diagram and skeleton extraction

In the next section we show in simulation the skeletons extracted on different maps using an area voronoi diagram or a line voronoi diagram. Our purpose is to use these skeletons as maximum-clearance paths for robotic navigation.

6 Experiments

First, we present the simulator used for our experiments. Then, we show the different cases of thick and thin bisector extractions. And finally, we give the emergent skeleton computed on different maps using an area voronoi diagram or a line voronoi diagram. Some of the maps used in simulation are taken from [1, 9] and others were created especially for this work. These maps expose the drawbacks of area voronoi skeleton regarding robotic navigation. Also, it shows that thinner line skeletons are interesting when it comes to narrow passages.

6.1 Simulator

We use the simulator SmartTiles (project gforge.inria.fr) for our experiments, it simulates a network of intelligent tiles. It was developped and is still maintained by Olivier Rochel from the SED Engineers Team. The rules for identification, propagation and skeleton extraction presented in this report were implemented in Python for synchronous and asynchronous modes. In the next section, we present the different cases of bisector extraction using thick extraction or thin extraction.

6.2 Bisectors

We present the results of Sk for thick (11) or thin (12) bisector emergence.

The bisectors given in Fig.8 were obtained by applying the first rule of extraction which selects the boundary cells of a Voronoi region. As we mention earlier in this report, the bisectors computed with this rule are thick. It has motivated us to develop a rule for extracting thinner skeletons.

Now we use the second extraction rule, which isolates thin bisectors by differenciating strict equidistance and modulo 1-equidistance from the sources. When a cell has a different identifier from one of its neighbors, it emerges as a boudary cell. Thin bisectors are colored in white, and their 4-connexity dilatation in yellow on Fig.9. The bisector extracted for each case are complete and have the minimum thickness allowed in our model.

We give more details about the spacing choices between two sources a and b. Let $s_x = |x_a - x_b|$ and $s_y = |y_a - y_b|$. If $s_x = s_y$ we have two cases, s_x is an odd or even distance. On the other hand, if $s_x \neq s_y$ we have $|\{odd, even\}^2| = 4$ more cases.

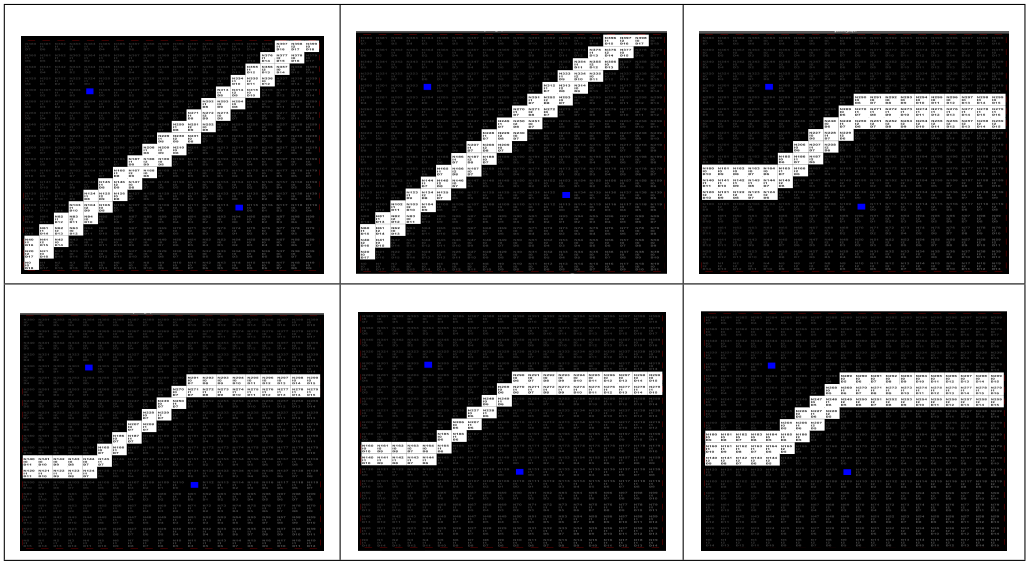


Figure 8: Thick Bisector Extraction Cases

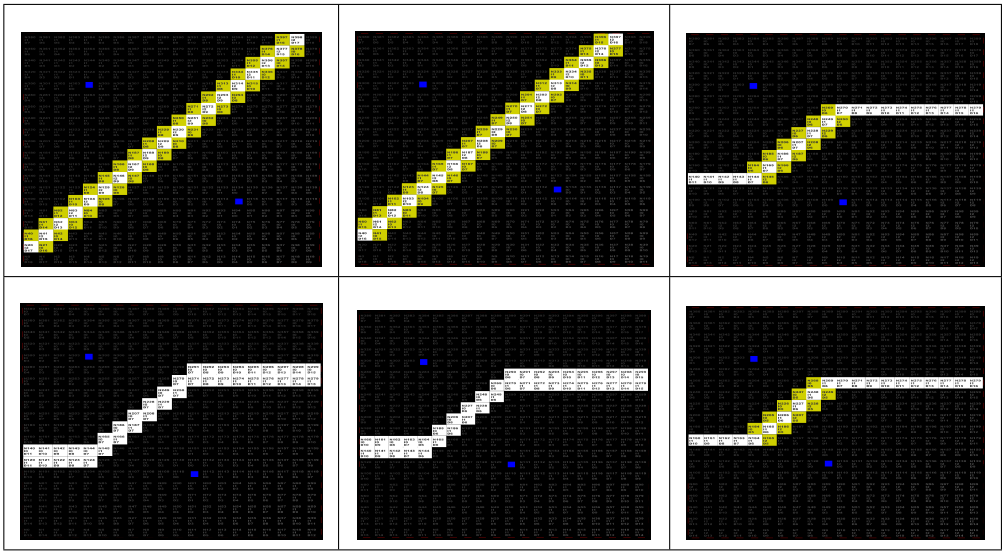


Figure 9: Thin Bisector Extraction Cases

6.3 Area Skeleton

In this section, we present our results in simulation regarding skeleton/navigation roads extraction using area Voronoi diagrams.

Fig.10 represents the thick skeleton associated to area Voronoi diagrams for different maps. Maps a and d exhibit no skeleton at all, this is explained by the fact that only one contiguous area is identified for these maps. On the opposite, the other maps produce a diagram and the extracted skeleton is shown in white. Two areas were computed for c and e, 9 for b and 13 for f. As pointed earlier, the skeleton extracted in map e does not enter concave shapes. For an application in robotic navigation, no roads are available for 2 of the environments, and even when provided they cannot enter concave shapes.

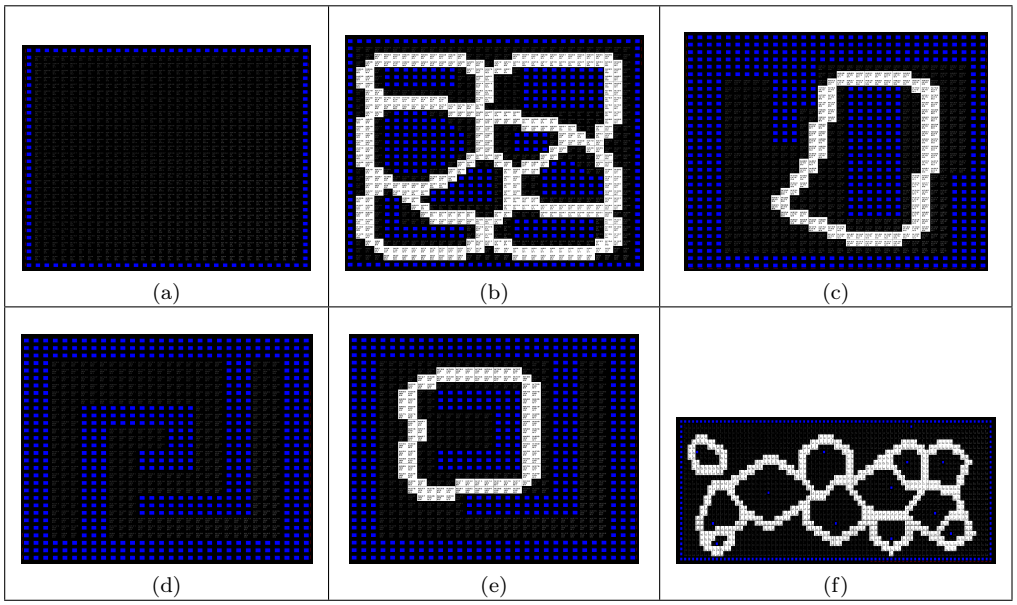


Figure 10: Thick Skeleton for Area Voronoi Diagram

6.4 Line Skeleton

In this part, we give the obtained skeleton extracted using line Voronoi diagrams on the same maps.

We can observe thick skeletons extracted from line Voronoi diagrams in Fig.11. Roads are available for each map and also enter concave shapes. These first aspects are satisfying but the extracted skeleton is too thick. For instance in maps b, d and e, the roads are close to obstacles. Therefore, we also provide the thinner skeletons in Fig.12, we can consider these skeletons as maximum clearance roadmaps.

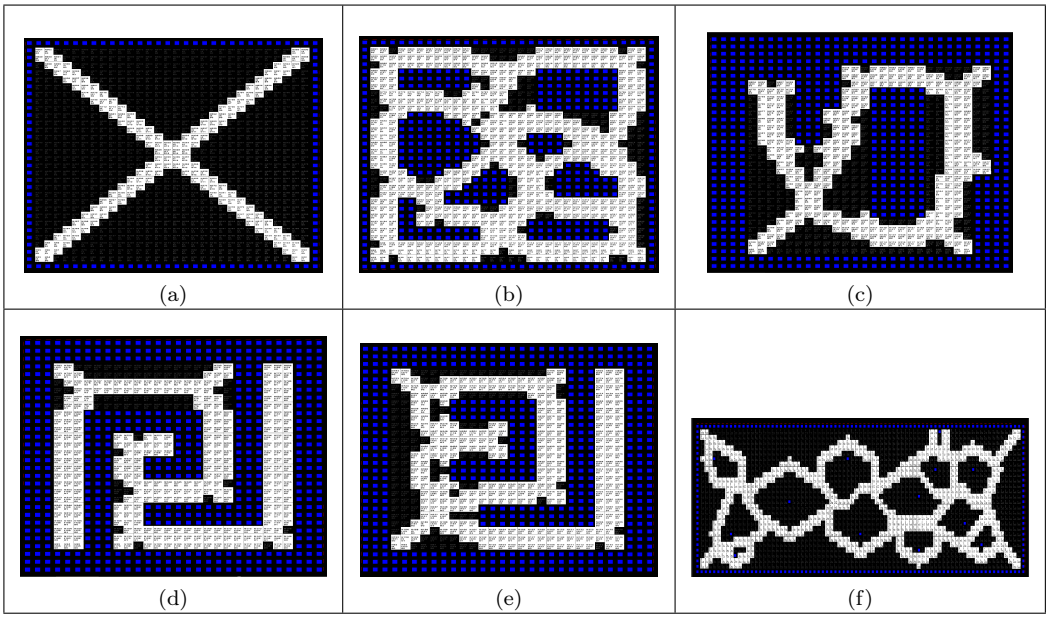


Figure 11: Thick Skeleton for Line Voronoi Diagram

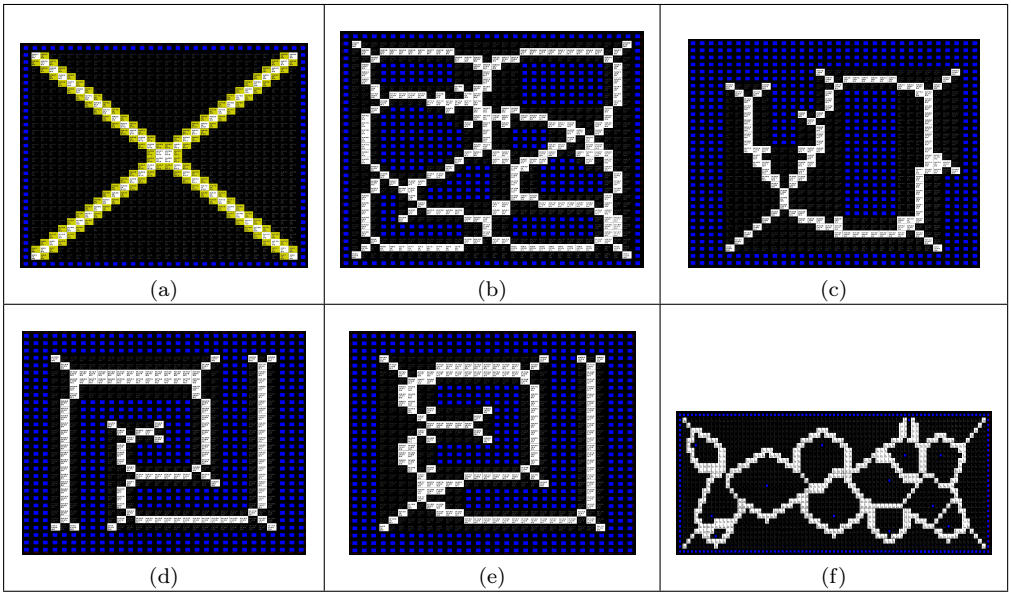


Figure 12: Thin Skeleton for Line Voronoi Diagram

7 Conclusion

This section summarizes the quality criteria satisfied by the skeleton extraction methods presented in this report. Then we discuss perspectives regarding robotic navigation using line Vononoi skeleton on smart floors.

7.1 Qualitative comparison of the different methods

Our method for skeleton extraction *LineV D*2 satisfies the criterion of synchronous/asynchronous calculus on a static or dynamic map (moving obstacles). Moreover, dilating the skeleton also allows to satisfy the criterion of 4-connexity in certain cases. Here is a table comparing the methods depending on qualitative criteria.

	Sync.	Async.	Centr.	Decentr.	Stat.	Dynam.	4-Conn.
LineVD2	Ok	Ok	Ok	Ok	Ok	Ok	Ok
Barraquand [2]	Ok	-	Ok	-	Ok	-	?
Tzionas [9]	Ok	-	-	Ok	Ok	-	-
Adamatzky [1]	Ok	-	-	Ok	Ok	-	-

7.2 Perspectives

In this report, we have shown how dynamic roadmaps can be extracted asynchronously in a decentralized fashion using a network of floor sensors. We can mention an application of robotic navigation using a floor equipped with smart tiles [7], in which each tile has its own sensors, processing unit and memory. Applying our work in such ambient intelligent environments can represent an interesting realization for robotic navigation.

References

[1] a.I. Adamatzky. Voronoi-like partition of lattice in cellular automata. *Mathematical and Computer Modelling*, 23(4):51–66, February 1996.

[2] J Barraquand, B Langlois, and J C Latombe. Numerical potential field techniques for robot path planning, 1991.

[3] Harry Blum. A Transformation for Extracting New Descriptors of Shape. *Models for the Perception of Speech and Visual Form*, pages 362–380, 1967.

[4] F. Boekhorst. Ambient intelligence, the next paradigm for consumer electronics: how will it affect silicon? *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, 1:28–31, 2002.

[5] Luidnel Maignan and Frederic Gruau. Convex Hulls on Cellular Spaces: Spatial Computing on Cellular Automata. *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 67–72, October 2011.

[6] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[7] Nicolas Pépin, Olivier Simonin, and François Charpillet. Intelligent tiles - putting situated multi-agents models in real world. In *ICAART*, pages 513–519, 2009.

[8] E. Thiel. *Les distances de chanfrein en analyse d'images : fondements et applications*. Thèse de Doctorat, Université Joseph Fourier, Grenoble 1, Sept 1994. `http://pageperso.lif.univ-mrs.fr/~edouard.thiel/these` .

[9] Panagiotis G Tzionas, Adonios Thanailakis, and Philippos G Tsalides. Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata. *IEEE Transactions on Robotics*, 13(2):237–250, 1997.

Appendix

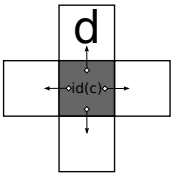
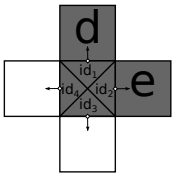
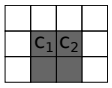
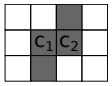
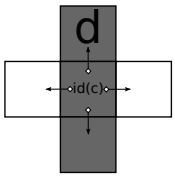
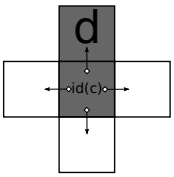
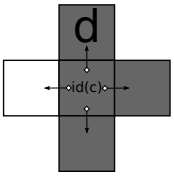
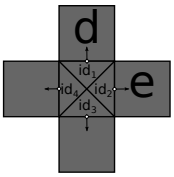
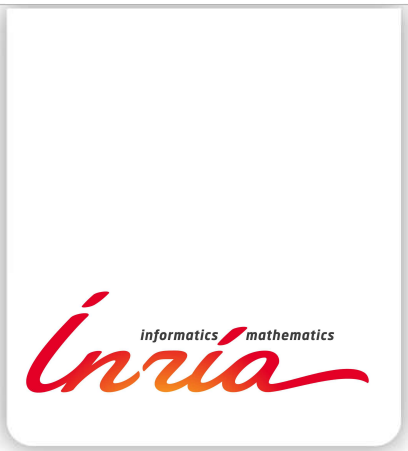
Class	Id presented to neighbor d
 (a) $Class_I$	$Eid_I(c,d) : c.id \leftarrow c.id$
 (b) $Class_{II}$, 4 ids  case1  case2	$Eid_{II}(c,d) : c.id \leftarrow \begin{cases} c.id \cup d.id & if\ d.cl = II \wedge case1(c,d) \\ e.id & if\ d.cl = II \wedge case2(c,e) \\ d.id & if\ d.cl \in \{III, IV, V\} \\ e.id & if\ d.cl = VI \end{cases}$
 (e) $Class_{III}$  (f) $Class_{IV}$	$Eid_{III/IV}(c,d) : c.id \leftarrow \begin{cases} c.id & if\ d.cl = II \\ c.id \cup d.id & if\ d.cl \in \{III, IV\} \\ c.id & if\ d.cl \in \{V, VI\} \end{cases}$
 (g) $Class_V$	$Eid_V(c,d) : c.id \leftarrow \begin{cases} c.id \cup d.id & if\ d.cl = c.cl, \\ c.id & else. \end{cases}$
 (h) $Class_{VI}$, 4 ids	$Eid_{VI}(c,d) : c.id \leftarrow \begin{cases} e.id' & if\ d.cl = II \wedge e.cl = II \\ d.id' & if\ d.cl = II \wedge e.cl \in \{III, IV, V\} \\ d.id & if\ d.cl \in \{III, IV, V\} \\ c.id & if\ d.cl = VI \end{cases}$

Figure 13: Edge Identification Consensus

Contents



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399